Apple Assembly Line

$1.50

Volume 4 -- Issue 2 November, 1983

In This Issue...

Tearing into ProDOS

Have we got a treat for you! You've heard about ProDOS, the new operating system for the Apple II's. Its main advantage over DOS 3.3 is speed, and on the next page of this issue you'll start to see what makes it so fast. ProDOS uses a completely different technique for translating between memory bytes and nibble-coded disk data, and here it is! Start reading Bob's completely commented disassembly.

Holiday Special Prices

Remember that we are offering special prices on several popular products from our list. Check the ad on page two for details. We are also having a sale on back issues of Apple Assembly Line: now only $1.00 each, rather than the usual $1.50. This is the time to complete your set! Subscription rates will be going up as of the first of the year, but you can still renew at the current prices. Let us hear from you.

Non-volatile RAM

Rodney Jacks, a Mostek engineer, tells us of a very interesting new chip: a 2K-byte static RAM, plug compatible with a 2716 EPROM, with a built-in lithium battery. Call your distributor and ask for Mostek MK48Z02. I can hardly wait to get some.

Commented Listing of ProDOS $F800-$F90B, $F996-FEBD

....................Bob Sander-Cederlof

ProDOS boots its bulk into the RAM card, from $D000 thru $FFFF. More is loaded into the alternate $D000-DFFF space, and all but 255 bytes are reserved out of the entire 16K space.

A system global page is maintained from $BF00-BFFF, for various variables and linkage routines. All communication between machine language programs and ProDOS is supposed to be through MLI (Machine Language Interface) calls and the system global page.

One of the first things I did with ProDOS was to start dis-assembling and commenting it. I want to know what is inside and how it works! Apple's 4-inch thick binder tells a lot, but not all.

Right away I ran into a roadblock: to disassemble out of the RAM card it has to be turned on. There is no monitor in the RAM card when ProDOS is loaded. Turning on the RAM card from the motherboard monitor causes a loud crash!

I overcame most of the problem by copying a monitor into the $F800-FFFF region of the RAM card like this:

```
        *C089 C089 F800<F800.FFFFM
        *C083 C083
```

The double C089 write-enables the RAM card, while memory reads are still from the motherboard. The rest of the line copies a monitor up. The two C083's get me into the RAM card monitor, ready to type things like "D000LLLLLLLLLLL"

But what about dis-assemblies of the space between $F800 and $FFFF? For this I had to write a little move program. My program turned on the RAM card and copied $F800-FFFF down to $6800-6FFF. Then I BSAVEd it, and later disassembled it.

The code from $F800-FFFF is mostly equivalent to what is in DOS 3.3 from $B800-BFFF. First I found a read/write block subroutine, which calls an RWTS-like subroutine twice per block. (All ProDOS works with 512-byte blocks, rather than sectors; this is like Apple Pascal, and the Apple ///.)

The listing which follows shows the RWB and RWTS subroutines, along with the READ.ADDRESS and READ.SECTOR subroutines. Next month I plan to lay out the SEEK.TRACK and WRITE.SECTOR subroutines, as well as the interrupt and reset handling code.

The outstanding difference between ProDOS and DOS 3.3 disk I/O is speed. ProDOS is considerably faster. Most of the speed increase is due to handling the conversion between memory-bytes and disk-bytes on the fly. DOS pre-converted a 256-byte block into 342 bytes in a special buffer, and then wrote the 342 bytes; ProDOS forms the first 86 bytes of the disk data in a special buffer, writes them, and then proceeds to write the rest of the data directly from the caller's buffer. When

S-C Macro Assembler Version 1.0		$80.00
S-C Macro Assembler Version 1.1 Update		$12.50
Full Screen Editor for S-C Macro Assembler Includes complete source code.	(reg. $49.00)	$40.00**
S-C Cross Reference Utility		$20.00
S-C Cross Reference Utility with Complete Source Code		$50.00
DISASM Dis-Assembler (RAK-Ware)		$30.00
Quick-Trace (Anthro-Digital)	(reg. $50.00)	$45.00
The Visible Computer: 6502 (Software Masters)	(reg. $50.00)	$40.00**

S-C Word Processor (the one we use!) With fully commented source code.		$50.00
Applesoft Source Code on Disk Very heavily commented. Requires Applesoft and S-C Assembler.		$50.00
ES-CAPE: Extended S-C Applesoft Program Editor	(reg. $60.00)	$40.00**

AAL Quarterly Disks	each $15.00

Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.

QD#1: Oct-Dec 1980	QD#2: Jan-Mar 1981	QD#3: Apr-Jun 1981
QD#4: Jul-Sep 1981	QD#5: Oct-Dec 1981	QD#6: Jan-Mar 1982
QD#7: Apr-Jun 1982	QD#8: Jul-Sep 1982	QD#9: Oct-Dec 1982
QD#10: Jan-Mar 1983	QD#11: Apr-Jun 1983	QD#12: Jul-Sep 1983

Double Precision Floating Pcint for Applesoft Provides 21-digit precision for Applesoft programs. Includes sample Applesoft subroutines for standard math functions.		$50.00
Amper-Magic (Anthro-Digital)	(reg. $75.00)	$67.50
Amper-Magic Volume 2 (Anthro-Digital)	(reg. $35.00)	$30.00
Routine Machine (Southwestern Data Systems)	(reg. $64.95)	$60.00
FLASH! Integer BASIC Compiler (Laumer Research)	(reg. $79.00)	$50.00**

Blank Diskettes (Premium quality, single-sided, double density, with hub rings)	package of 20 for $45.00
Vinyl disk pages, 6"x8.5", hold one disk each	10 for $6.00
Diskette Mailing Protectors	10-99: 40 cents each 100 or more: 25 cents each
ZIF Game Socket Extender	$20.00
Shift-Key Modifier	$15.00

Grappler+ Printer Interface (Orange Micro)	($175.00)	$150.00
Bufferboard 16K Buffer for Grappler (Orange Micro)	($175.00)	$150.00
Buffered Grappler+ NEW!! Interface and 16K Buffer	($239.00)	$200.00

Books, Books, Books........................compare our discount prices!

"The Apple][Circuit Description", Gayler	($22.95)	$21.00
"Enhancing Your Apple II, vol. 1", Lancaster	($17.95)	$17.00
"Incredible Secret Money Machine", Lancaster	($7.95)	$7.50
"Micro Cookbook, vol. 1", Lancaster	($15.95)	$15.00
"Micro Cookbook, vol. 2", Lancaster	($15.95)	$15.00
"Beneath Apple DOS", Worth & Lechner	($19.95)	$18.00
"Bag of Tricks", Worth & Lechner, with diskette	($39.95)	$36.00
"Apple Graphics & Arcade Game Design", Stanton	($19.95)	$18.00
"Assembly Lines: The Book", Roger Wagner	($19.95)	$18.00
"What's Where in the Apple", Second Edition	($24.95)	$23.00
"What's Where Guide" (updates first edition)	($9.95)	$9.00
"6502 Assembly Language Programming", Leventhal	($18.95)	$18.00
"6502 Subroutines", Leventhal	($17.95)	$17.00

Add $1.50 per book for US postage. Foreign orders add postage needed.

Whatever Else You Need..........................Call for Our Low Prices

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We accept Master Card, VISA and American Express ***

(** Special price to subscribers only through December 31, 1983.)

reading, DOS read the 342 disk-bytes into a buffer for later decoding into the caller's buffer. ProDOS reads and decodes simultaneously directly into the caller's buffer. This is achieved by extensive use of tables and self-modifying code.

Not only is direct time saved by doing a lot less copying of buffers, but also the sector interleaving can be arranged so that only two revolutions are required to read all 8 blocks on a track.

I believe Apple Pascal uses the same technique, at least for reading.

Whoever coded ProDOS decided to hard-code some parameters which DOS used to keep in tables specified by the user. For example, the number which tells how long to wait for a drive motor to rev up used to be kept in a Device Characteristics Table (DCT). That value is now inside a "LDA #$E8" instruction at $F84F. That means that if you are using a faster drive you have to figure out how to patch and unpatch ProDOS to take advantage of it.

Another hard-coded parameter is the maximum block number. This is no longer part of the data on an initialized disk. It is now locked into the four instructions at $F807-F80D, at a maximum of 279. If you have a 40- or 70-track drive, you can only use 35. Speaking of tracks, the delay tables for track seeking are still used, but they are of course buried in this same almost-unreachable area. If you have a drive with faster track-to-track stepping, the table to change is at $FB73-FB84.

Calls to RWTS in DOS 3.3 involved setting up two tables, an IOB and a DCT. The IOB contained all the data about slot, drive, track, sector, buffer address, etc. The DCT was a 5-byte table with data concerning the drive. ProDOS RWB is called in an entirely different way. A fixed-position table located at $42-47 in page zero is set up with the command, slot, buffer address, and block number.

There are three valid commands, which I call test, read, and write. Test (0) starts up the indicated drive. If it is successful, a normal return occurs; if not, you get an error return (carry set, and (A) non-zero). Read (1) and write (2) are what you expect them to be. RWB has a very simple job: validate the call parameters in $42-47, convert block number to track and sector, and call RWTS twice (once for each sector of the block).

ProDOS RWTS expects the sector number in the A-register, and the track in a variable at $FB56. RWTS handles turning on the drive motor and waiting for it to come up to speed. RWTS then calls SEEK.TRACK to find the desired track, READ.ADDRESS to find the selected sector, and branches to READ.SECTOR or WRITE.SECTOR depending on the command.

READ.ADDRESS is virtually the same in ProDOS as it was in DOS 3.3. READ.SECTOR is entirely different. I should point out here that ProDOS diskettes are entirely compatible with Apple

/// diskettes. The file structures are exactly the same. Both ProDOS and Apple /// diskettes use the same basic recording techniques on the disk as DOS 3.3, so the diskettes are copyable using standard DOS 3.3 copiers such as the COPYA program on your old System Master Diskette.

READ.SECTOR begins by computing several addresses and plugging them into the code further down. (This enables the use of faster addressing modes, saving enough cycles to leave time for complete decoding of disk data on the fly.) First the disk slot number is merged into the instructions which read bytes from the drive. Next the caller's buffer address is put into the store instructions.

Note that the byte from the disk is loaded into the X-register, then used to index into BYTE.TABLE, at $F996, to get the equivalent 6-bit data value. Since a disk byte may only have certain values, there is some space within BYTE.TABLE that will never be accessed. Most of this unused space contains $FF bytes, but some of it is used for other small tables: BIT.PAIR.LEFT, .MIDDLE, and .RIGHT, and DATA.TRAILER. These are used by WRITE.SECTOR, which we'll look at next month.

Your buffer is divided into three parts: two 86-byte chunks, and one of 84 bytes. Data coming from the disk is in four chunks: three of 86 bytes, and one of 84.

The first chunk contains the lower two bits from every byte in the original data. READ.SECTOR reads this chunk into TBUF, so that the bits will be available later for merging with the upper six of each byte. ($FC53-FC68)

The second chunk contains the upper six bits from the first 86 bytes of the original data. $FC69-FC83 reads the chunk and merges in the lower two bits from TBUF, storing the completed bytes in the first 85 bytes of the caller's buffer. The last (86th) byte is saved on the stack (I am not sure why), and not stored in the caller's buffer until after all the rest of the data has been read.

A tricky manipulation is necessary to merge in those lower two bits. The data in TBUF has those bits in backward order, packed together with the bits from the other chunks. There was a good diagram of this on page 10 of the June 1981 issue of AAL. DOS merged them with a complex time-consuming shifting process. ProDOS does a swift table lookup, using the TBUF byte as an index to the BIT.PAIR.TABLE.

BIT.PAIR.TABLE has four bytes per row. The first three in each row supply the bit pairs; the fourth is used by SECTOR.WRITE to encode data, and will be covered next month.

When $FC69-FC83 is reading the first chunk, the first byte in each row is used to supply the lower two data bits. The byte in TBUF corresponding to the current position in the chunk selects a byte from BIT.PAIR.TABLE, and the two parts are merged together.

The next two chunks are handled just like the one I just described. After all the data has been read, READ.SECTOR expects to have accumulated a checksum of 00, and expects to find a trailing $EB after the data. Return with carry clear indicates all went well; carry set indicates a read error (bad checksum, missing header, or missing trailer).

I can't help wondering: can this fast read technique be fit into DOS 3.3? It takes a little more code and table space, but on the other hand it uses 256 bytes less of intermediate buffer. If we sacrificed the INIT command, could both the fast read and write be squeezed into DOS 3.3?

```
                  1010 *SAVE S.PRODOS F800-FFFF
                  1020 *--------------------------------
003A-             1030 RUNNING.SUM .EQ $3A
003A-             1040 TBUF.0      .EQ $3A
003B-             1050 Z.3B        .EQ $3B
003C-             1060 Z.3C        .EQ $3C
003D-             1070 Z.3D        .EQ $3D
003E-             1080 SLOT.X16    .EQ $3E
003F-             1090 Z.3F        .EQ $3F
                  1100 *--------------------------------
0042-             1110 RWB.COMMAND .EQ $42
0043-             1120 RWB.SLOT    .EQ $43   DSSSXXXX
0044-             1130 RWB.BUFFER  .EQ $44,45
0046-             1140 RWB.BLOCK   .EQ $46,47   0...279
                  1150 *--------------------------------
4700-             1160 BUFF.BASE .EQ $4700 DUMMY ADDRESS FOR ASSEMBLY ONLY
                  1170 *--------------------------------
BF56-             1180 SAVE.LOC45 .EQ $BF56
BF57-             1190 SAVE.D000  .EQ $BF57
BF88-             1200 INTAREG    .EQ $BF88
BF8D-             1210 INTBANKID  .EQ $BF8D
BFD3-             1220 IRQXIT.3   .EQ $BFD3
                  1230 *--------------------------------
C080-             1240 DRV.PHASE  .EQ $C080
C088-             1250 DRV.MTROFF .EQ $C088
C089-             1260 DRV.MTRON  .EQ $C089
C08A-             1270 DRV.ENBL.0 .EQ $C08A
C08C-             1280 DRV.Q6L    .EQ $C08C
C08D-             1290 DRV.Q6H    .EQ $C08D
C08E-             1300 DRV.Q7L    .EQ $C08E
C08F-             1310 DRV.Q7H    .EQ $C08F
                  1320 *--------------------------------
                  1330 *                 <<<COMPUTED >>>
0060-             1340 MODIFIER .EQ $60  <<<SLOT * 16>>>
                  1350 *--------------------------------
                  1360        .OR $F800
                  1370        .TA $800
                  1380 *--------------------------------
                  1390 *      READ/WRITE A BLOCK
                  1400 *
                  1410 *      1.  ASSURE VALID BLOCK NUMBER (0...279)
                  1420 *      2.  CONVERT BLOCK NUMBER TO TRACK/SECTOR
                  1430 *          TRACK = INT(BLOCK/8)
                  1440 *          BLOCK   SECTORS
                  1450 *          -----   -------
                  1460 *            0      0 AND 2
                  1470 *            1      4 AND 6
                  1480 *            2      8 AND 10
                  1490 *            3     12 AND 14
                  1500 *            4      1 AND 3
                  1510 *            5      5 AND 7
                  1520 *            6      9 AND 11
                  1530 *            7     13 AND 15
                  1540 *      3.  CALL RWTS TWICE
                  1550 *      4.  RETURN WITH ERROR STATUS
                  1560 *--------------------------------
                  1570 RWB
F800- A5 46       1580        LDA RWB.BLOCK     BLOCK MUST BE 0...279
F802- A6 47       1590        LDX RWB.BLOCK+1
F804- 8E 56 FB    1600        STX RWTS.TRACK
F807- F0 07       1610        BEQ .1        ...BLOCK # LESS THAN 256
F809- CA          1620        DEX
```

QUICKTRACE

relocatable program traces and displays the actual machine operations, *while* it is running without interfering with those operations. Look at these *FEATURES:*

***Single-Step** mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

***Trace** mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

***Background** mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

***QUICKTRACE** allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

***Two optional display formats** can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

***QUICKTRACE** is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

***QUICKTRACE** is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

***QUICKTRACE** is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while **QUICKTRACE** is alive.*

***QUICKTRACE** is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

QUICKTRACE *$50*

Is a trademark of Anthro-Digital, Inc.

Copyright © 1981
Written by John Rogers

See these programs at participating Computerland and other fine computer stores.

Anthro - Digital Software, Inc.
P.O. Box 1385 Pittsfield, MA 01202

```
F80A- D0 2A    1630        BNE .5       ...BLOCK # MORE THAN 511
F80C- C9 18    1640        CMP #$18
F80E- B0 26    1650        BCS .5       ...BLOCK # MORE THAN 279
F810- A0 05    1660 .1     LDY #5       SHIFT 5 BITS OF TRACK #
F812- 0A       1670 .2     ASL              RWTS.TRACK  A-REG
F813- 2E 56 FB 1680        ROL RWTS.TRACK   ----------  --------
F816- 88       1690        DEY              00TTTTTT    ABC00000
F817- D0 F9    1700        BNE .2
F819- 0A       1710        ASL          TRANSFORM BLOCK # INTO SECTOR #
F81A- 90 02    1720        BCC .3       ABC00000 --> 0000BC0A
F81C- 09 10    1730        ORA #$10
F81E- 4A       1740 .3     LSR
F81F- 4A       1750        LSR
F820- 4A       1760        LSR
F821- 4A       1770        LSR
F822- 48       1780        PHA
F823- 20 3A F8 1790        JSR RWTS     R/W FIRST SECTOR OF BLOCK
F826- 68       1800        PLA
F827- B0 09    1810        BCS .4       ...ERROR
F829- E6 45    1820        INC RWB.BUFFER+1
F82B- 69 02    1830        ADC #2
F82D- 20 3A F8 1840        JSR RWTS     R/W SECOND SECTOR OF BLOCK
F830- C6 45    1850        DEC RWB.BUFFER+1
F832- AD 58 FB 1860 .4     LDA RWTS.ERROR
F835- 60       1870        RTS
               1880 *---BLOCK NUMBER > 279-----------
F836- A9 27    1890 .5     LDA #$27     I/O ERROR
F838- 38       1900        SEC
F839- 60       1910        RTS
               1920 *--------------------------------
               1930 *      READ/WRITE A GIVEN SECTOR
               1940 *--------------------------------
               1950 RWTS
F83A- A0 01    1960        LDY #1       TRY SEEKING TWICE
F83C- 8C 6A FB 1970        STY SEEK.COUNT
F83F- 8D 57 FB 1980        STA RWTS.SECTOR
F842- A5 43    1990        LDA RWB.SLOT
F844- 29 70    2000        AND #$70     0SSS0000
F846- 85 3E    2010        STA SLOT.X16
F848- 20 9B FE 2020        JSR WAIT.FOR.OLD.MOTOR.TO.STOP
F84B- 20 DA FC 2030        JSR CHECK.IF.MOTOR.RUNNING
F84E- 08       2040        PHP          SAVE ANSWER (.NE. IF RUNNING)
F84F- A9 E8    2050        LDA #$E8     MOTOR STARTING TIME
F851- 8D 70 FB 2060        STA MOTOR.TIME+1 ONLY HI-BYTE NECESSARY
F854- A5 43    2070        LDA RWB.SLOT     SAME SLOT AND DRIVE?
F856- CD 59 FB 2080        CMP OLD.SLOT
F859- 8D 59 FB 2090        STA OLD.SLOT
F85C- 08       2100        PHP              SAVE ANSWER
F85D- 0A       2110        ASL              DRIVE # TO C-BIT
F85E- BD 89 C0 2120        LDA DRV.MTRON,X  START MOTOR
F861- 90 01    2130        BCC .1           ...DRIVE 0
F863- E8       2140        INX              ...DRIVE 1
F864- BD 8A C0 2150 .1     LDA DRV.ENBL.0,X ENABLE DRIVE X
F867- 28       2160        PLP              SAME SLOT/DRIVE?
F868- F0 0A    2170        BEQ .3           ...YES
F86A- 28       2180        PLP          DISCARD ANSWER ABOUT MOTOR GOING
F86B- A0 07    2190        LDY #7       DELAY 150-175 MILLISECS
F86D- 20 85 FB 2200 .2     JSR DELAY.100    DELAY 25 MILLISECS
F870- 88       2210        DEY
F871- D0 FA    2220        BNE .2
F873- 08       2230        PHP          SAY MOTOR NOT ALREADY GOING
F874- A5 42    2240 .3     LDA RWB.COMMAND  0=TEST, 1=READ, 2=WRITE
F876- F0 06    2250        BEQ .4           ...0, MERELY TEST
F878- AD 56 FB 2260        LDA RWTS.TRACK
F87B- 20 0C F9 2270        JSR SEEK.TRACK
F87E- 28       2280 .4     PLP          WAS MOTOR ALREADY GOING?
F87F- D0 0F    2290        BNE .6       ...YES
F881- A9 01    2300 .5     LDA #1       DELAY 100 USECS
F883- 20 85 FB 2310        JSR DELAY.100
F886- AD 70 FB 2320        LDA MOTOR.TIME+1
F889- 30 F6    2330        BMI .5       ...WAIT TILL IT OUGHT TO BE
F88B- 20 DA FC 2340        JSR CHECK.IF.MOTOR.RUNNING
F88E- F0 5C    2350        BEQ .14      ...NOT RUNNING YET, ERROR
F890- A5 42    2360 .6     LDA RWB.COMMAND
F892- F0 69    2370        BEQ .17      CHECK IF WRITE PROTECTED
F894- 4A       2380        LSR          .CS. IF READ, .CC. IF WRITE
F895- B0 03    2390        BCS .7       ...READ
F897- 20 F0 FD 2400        JSR PRE.NYBBLE   ...WRITE
F89A- A0 40    2410 .7     LDY #64      TRY 64 TIMES TO FIND THE SECTOR
```

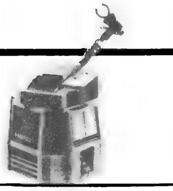
```
F89C- 8C 69 FB 2420         STY SEARCH.COUNT
F89F- A6 3E    2430 .8      LDX SLOT.X16
F8A1- 20 98 FB 2440         JSR READ.ADDRESS
F8A4- 90 1A    2450         BCC .10        ...FOUND IT
F8A6- CE 69 FB 2460 .9      DEC SEARCH.COUNT
F8A9- 10 F4    2470         BPL .8         ...KEEP LOOKING
F8AB- A9 27    2480         LDA #$27      I/O ERROR CODE
F8AD- CE 6A FB 2490         DEC SEEK.COUNT   ANY TRIES LEFT?
F8B0- D0 3A    2500         BNE .14        ...NO, I/O ERROR
F8B2- AD 5A FB 2510         LDA CURRENT.TRACK
F8B5- 48       2520         PHA
F8B6- 0A       2530         ASL            SLIGHT RE-CALIBRATION
F8B7- 69 10    2540         ADC #$10
F8B9- A0 40    2550         LDY #64        ANOTHER 64 TRIES
F8BB- 8C 69 FB 2560         STY SEARCH.COUNT
F8BE- D0 0E    2570         BNE .11        ...ALWAYS
F8C0- AC 6F FB 2580 .10     LDY HDR.TRACK     ACTUAL TRACK FOUND
F8C3- CC 5A FB 2590         CPY CURRENT.TRACK
F8C6- F0 0F    2600         BEQ .12           FOUND THE RIGHT ONE
F8C8- AD 5A FB 2610         LDA CURRENT.TRACK WRONG ONE, TRY AGAIN
F8CB- 48       2620         PHA
F8CC- 98       2630         TYA            STARTING FROM TRACK FOUND
F8CD- 0A       2640         ASL
F8CE- 20 D3 FC 2650 .11     JSR UPDATE.TRACK.TABLE
F8D1- 68       2660         PLA
F8D2- 20 0C F9 2670         JSR SEEK.TRACK
F8D5- 90 C8    2680         BCC .8         ...ALWAYS
F8D7- AD 6E FB 2690 .12     LDA HDR.SECTOR
F8DA- CD 57 FB 2700         CMP RWTS.SECTOR
F8DD- D0 C7    2710         BNE .9
F8DF- A5 42    2720         LDA RWB.COMMAND
F8E1- 4A       2730         LSR
F8E2- 90 10    2740         BCC .15           ...WRITE
F8E4- 20 FD FB 2750         JSR READ.SECTOR   ...READ
F8E7- B0 BD    2760         BCS .9            ...READ ERROR
F8E9- A9 00    2770 .13     LDA #0         NO ERROR
F8EB- D0       2780         .HS D0         "BNE"...NEVER, JUST SKIPS "SEC"
F8EC- 38       2790 .14     SEC            ERROR
F8ED- 8D 58 FB 2800         STA RWTS.ERROR    SAVE ERROR CODE
F8F0- BD 88 C0 2810         LDA DRV.MTROFF,X  STOP MOTOR
F8F3- 60       2820         RTS               RETURN
               2830 *-------------------------------
F8F4- 20 00 FD 2840 .15     JSR WRITE.SECTOR
F8F7- 90 F0    2850 .16     BCC .13        ...NO ERROR
F8F9- A9 2B    2860         LDA #$2B       WRITE PROTECTED ERROR CODE
F8FB- D0 EF    2870         BNE .14        ...ALWAYS
F8FD- A6 3E    2880 .17     LDX SLOT.X16 CHECK IF WRITE PROTECTED
F8FF- BD 8D C0 2890         LDA DRV.Q6H,X
F902- BD 8E C0 2900         LDA DRV.Q7L,X
F905- 2A       2910         ROL
F906- BD 8C C0 2920         LDA DRV.Q6L,X
F909- 4C F7 F8 2930         JMP .16        GIVE ERROR IF PROTECTED
               2940 *-------------------------------
```

SEEK.TRACK is in this gap.
The following tables start at $F996.

```
3660 *--------------------------------
3670 *      VALUE READ FROM DISK IS INDEX INTO THIS TABLE
3680 *      TABLE ENTRY GIVES TOP 6 BITS OF ACTUAL DATA
3690 *
3700 *      OTHER DATA TABLES ARE IMBEDDED IN THE UNUSED
3710 *      PORTIONS OF THE BYTE.TABLE
3720 *--------------------------------
3730 BYTE.TABLE .EQ *-$96
3740        .HS 0004FFFF080CFF101418
3750 BIT.PAIR.LEFT
3760        .HS 008040C0
3770        .HS FFFF1C20FFFFFF24282C
3780        .HS 3034FFFF383C4044
3790        .HS 494CFF5054585C606468
3800 BIT.PAIR.MIDDLE
3810        .HS 00201030
3820 DATA.TRAILER
3830        .HS DEAAEBFF
3840        .HS FFFFFF6CFF70
3850        .HS 7478FFFFFF7CFFFF
3860        .HS 8084FF888C9094989CA0
3870 BIT.PAIR.RIGHT
3880        .HS 0008040C
3890        .HS FFA4A8ACFFB0B4B8BCC0
3900        .HS C4C8FFFFCCD0D4D8
3910        .HS DCE0FFE4E8ECF0F4
3920        .HS F8FC
```

```
3930 *--------------------------------
3940 BIT.PAIR.TABLE
3950        .HS 00000096
3960        .HS 02000097
3970        .HS 0100009A
3980        .HS 0300009B
3990        .HS 0002009D
4000        .HS 0202009E
4010        .HS 0102009F
4020        .HS 030200A6
4030        .HS 000100A7
4040        .HS 020100AB
4050        .HS 010100AC
4060        .HS 030100AD
4070        .HS 000300AE
4080        .HS 020300AF
4090        .HS 010300B2
4100        .HS 030300B3
4110        .HS 000002B4
4120        .HS 020002B5
4130        .HS 010002B6
4140        .HS 030002B7
4150        .HS 000202B9
4160        .HS 020202BA
4170        .HS 010202BB
4180        .HS 030202BC
4190        .HS 000102BD
4200        .HS 020102BE
4210        .HS 010102BF
4220        .HS 030102CB
4230        .HS 000302CD
4240        .HS 020302CE
4250        .HS 010302CF
4260        .HS 030302D3
4270        .HS 000001D6
4280        .HS 020001D7
4290        .HS 010001D9
4300        .HS 030001DA
4310        .HS 000201DB
4320        .HS 020201DC
4330        .HS 010201DD
4340        .HS 030201DE
4350        .HS 000101DF
4360        .HS 020101E5
4370        .HS 010101E6
4380        .HS 030101E7
4390        .HS 000301E9
4400        .HS 020301EA
4410        .HS 010301EB
4420        .HS 030301EC
4430        .HS 000003ED
4440        .HS 020003EE
4450        .HS 010003EF
4460        .HS 030003F2
4470        .HS 000203F3
4480        .HS 020203F4
4490        .HS 010203F5
4500        .HS 030203F6
4510        .HS 000103F7
4520        .HS 020103F9
4530        .HS 010103FA
4540        .HS 030103FB
4550        .HS 000303FC
4560        .HS 020303FD
4570        .HS 010303FE
4580        .HS 030303FF

                 4590 *--------------------------------
FB00-            4600 TBUF   .BS 86
                 4610 *--------------------------------
FB56- 07         4620 RWTS.TRACK    .HS 07
FB57- 0F         4630 RWTS.SECTOR   .HS 0F
FB58- 00         4640 RWTS.ERROR    .HS 00
FB59- 60         4650 OLD.SLOT      .HS 60
FB5A- 07         4660 CURRENT.TRACK .HS 07
FB5B- 00         4670               .HS 00
                 4680 *--------------------------------
FB58-            4690 OLD.TRACK.TABLE .EQ *-4
FB5C- 00 00      4700        .HS 0000     SLOT 2, DRIVE 0--DRIVE 1
FB5E- 00 00      4710        .HS 0000     SLOT 3
FB60- 00 00      4720        .HS 0000     SLOT 4
FB62- 00 00      4730        .HS 0000     SLOT 5
FB64- 0E 00      4740        .HS 0E00     SLOT 6
FB66- 00 00      4750        .HS 0000     SLOT 7
                 4760 *--------------------------------
FB68- 00         4770        .HS 00
                 4780 *--------------------------------
FB69-            4790 SEARCH.COUNT       .BS 1
FB6A-            4800 SEEK.COUNT         .BS 1
FB6B-            4810 STEP.CNT      .EQ *
FB6B-            4820 SEEK.D5.CNT   .EQ *
FB6B-            4830 X1X1X1X1           .BS 1   ALSO STEP.CNT & SEEK.D5.CNT
FB6C-            4840 CHECK.SUM          .BS 1
FB6D-            4850 HDR.CHKSUM         .BS 1
FB6E-            4860 HDR.SECTOR         .BS 1
FB6F-            4870 HDR.TRACK     .EQ *
FB6F-            4880 MOTOR.TIME         .BS 2   ALSO HDR.TRACK & HDR.VOLUME
FB71-            4890 CURRENT.TRACK.OLD .BS 1
FB72-            4900 TARGET.TRACK       .BS 1
                 4910 *--------------------------------
                 4920 *      DELAY TIMES FOR ACCELERATION & DECELERATION
                 4930 *      OF TRACK STEPPING MOTOR
                 4940 *--------------------------------
FB73- 01 30 28
FB76- 24 20 1E
FB79- 1D 1C 1C 4950 ONTBL  .HS 01302824201E1D1C1C
FB7C- 70 2C 26
FB7F- 22 1F 1E
FB82- 1D 1C 1C 4960 OFFTBL .HS 702C26221F1E1D1C1C
```



PERSONAL ROBOTS

Peripherals and Software for Personal Robots

NEW

VOICE COMMAND SYSTEM FOR HERO

MICROMATION proudly presents a new peripheral for HEATHKIT'S® HERO-I robot which elevates the robot to a new level of sophistication. We call this peripheral a Voice Command System (or VCS) because it not only consists of a voice recognizer, but also an advanced level machine language program for the robot which actually allows you to program robot movements by voice. We call the voice recognizer VOREC and the voice driven program VOCOL (VOice COmmand Language). Highlights of these two important parts of the VCS are described below.

VOREC

VOREC is a powerful, microprocessor controlled, speech recognition board which mounts next to, and interfaces with, our HERO-I MEMCOM BOARD. The recognizer has the following principal features and specifications:

- Speaker-dependent recognizer with nearly instantaneous word recognition rates.
- Recognition accuracy about 98%.
- Vocabulary of up to 256 words (stored as 16 word groups with 16 words in each group for greater recognition accuracy).
- 16K of onboard static RAM of which 14K is battery backed to retain recognized word parameters during power down.
- RS232 port for receiving commands from, and reporting status and words recognized to, the host (HERO).
- Highly sensitive audio input circuitry requires only an external speaker for audio input rather than a microphone. (This allows robot to receive commands from up to 15 feet away.)
- Utilizes state-of-the-art high speed (HC) CMOS chips and the new CMOS 65CO2 microprocessor for ultra low power consumption. Complete board consumes an incredibly low 45 ma while active and 1 ma when inactive.
- Speech recognition is accomplished by a software algorithm contained in a 2K EPROM. (Future product updates will require only replacement of this EPROM.)

VOCOL

This software is even more amazing than the voice recognition hardware. VOCOL is like a high level language for the robot (such as BASIC) which supports both deferred and immediate execution modes. The only difference is in BASIC you "write it," and in VOCOL you "speak it." The software is provided on an EPROM which plugs into a memory socket on our HERO-I MEMCOM BOARD. VOCOL has the following principal features:

- When first run, the robot talks to you through a voice training session in which you are asked to repeat words in his command vocabulary three (3) times.
- Following this training session, you can literally talk in a program of movements for later execution, or command immediate movement by voice.
- The robot prompts you for a command and when received, repeats it back to you for verification. If verified and if in immediate execution mode, the robot will execute the movement. If in deferred execution mode, the robot proceeds to write a machine language program in his memory for later execution. When your program of movements is complete, you signify this with a "STOP" command. A "GO" command will then cause the robot to execute the program it wrote in memory. After execution, the robot returns to the command mode.
- Complete instructions and installation manual.

The Voice Command System manual contains a complete description of how to use the VOREC board under program control from HERO. The 6808 Source Code for VOCOL is available on an APPLE® DOS 3.3 disk at additional cost. This source code is compatible with the SC-6800 CROSS ASSEMBLER.

VOCOL Source Code **$55.00** (not sold separately) **TOTAL SYSTEM PRICE $595.00**

NEW

POET

This is an Artificial Intelligence program similar in concept to STORYTELLER, but more advanced. The program uses an advanced self programming technique which allows the robot to speak self generated, random three line Haiku poems on an endless list of subjects. After HERO speaks a poem and likes it enough, he will make a comment about it or do some meaningful body movement.

PRICE: TAPE (machine code) **$20.00** **PRICE:** DISK (source code) **$30.00**

HERO MEMCOM BOARD

This product provides a means to develop programs for the robot using a personal computer, and expands the robot's memory with an additional 30K of RAM. This product includes:

- Two 8-bit bi-directional parallel ports with handshaking lines for superfast data transfers between the robot and a computer (connects directly to our APPLE-HERO COMMUNICATOR board), plus two 16-bit timers.
- An RS232 serial port for two-way communications between the robot and any computer having an RS232 serial port.
- Serial communications software in an onboard EPROM which allows uploading/downloading of programs via the serial port.
- Complete instruction manual and schematics.

PRICE $295.00

APPLE-HERO COMMUNICATOR

This product provides the hardware and software necessary to implement two way high speed parallel communication between an APPLE® computer and a HERO I robot equipped with our HERO MEMCOM BOARD. It includes:

- A peripheral card for an APPLE that contains two 8-bit parallel ports with handshaking lines, and two 16-bit timers.
- Data transfer software for the APPLE board and for the HERO MEMCOM BOARD burned into two 2716 EPROMS. These programs provide ultra fast two-way communications.
- A disk containing heavily commented 6808 and 6502 source codes for the communications software. These source codes are compatible with the S-C MACRO ASSEMBLER and the S-C 6800 CROSS ASSEMBLER available for the APPLE from the S-C SOFTWARE CORPORATION.

PRICE $159.00

Send check or money order to:

MICROMATION INC.

9104 Red Branch Rd.
Columbia, MD 21045

MasterCard VISA

Add $3.00 for shipping.

For information call:

(301) 730-1237

9 am-5pm Monday through Friday
MasterCard & Visa welcome

```
               4970 *--------------------------------
               4980 *      DELAY ABOUT 100*A MICROSECONDS
               4990 *      RUN DOWN MOTOR.TIME WHILE DELAYING
               5000 *--------------------------------
               5010 DELAY.100
FB85- A2 11    5020 .1     LDX #17
FB87- CA       5030 .2     DEX
FB88- D0 FD    5040        BNE .2
FB8A- EE 6F FB 5050        INC MOTOR.TIME
FB8D- D0 03    5060        BNE .3
FB8F- EE 70 FB 5070        INC MOTOR.TIME+1
FB92- 38       5080 .3     SEC
FB93- E9 01    5090        SBC #1
FB95- D0 EE    5100        BNE .1
FB97- 60       5110        RTS
               5120 *--------------------------------
               5130 READ.ADDRESS
FB98- A0 FC    5140        LDY #$FC     TRY 772 TIMES TO FIND $D5
FB9A- 8C 6B FB 5150        STY SEEK.D5.CNT         (FROM $FCFC TO $10000)
FB9D- C8       5160 .1     INY
FB9E- D0 05    5170        BNE .2       ...KEEP TRYING
FBA0- EE 6B FB 5180        INC SEEK.D5.CNT
FBA3- F0 56    5190        BEQ .11      ...THAT IS ENUF!
FBA5- BD 8C C0 5200 .2     LDA DRV.Q6L,X     GET NEXT BYTE
FBA8- 10 FB    5210        BPL .2
FBAA- C9 D5    5220 .3     CMP #$D5     IS IT $D5?
FBAC- D0 EF    5230        BNE .1       ...NO, TRY AGAIN
FBAE- EA       5240        NOP          ...YES, DELAY
FBAF- BD 8C C0 5250 .4     LDA DRV.Q6L,X     GET NEXT BYTE
FBB2- 10 FB    5260        BPL .4
FBB4- C9 AA    5270        CMP #$AA     NOW NEED $AA AND $96
FBB6- D0 F2    5280        BNE .3       ...NO, BACK TO $D5 SEARCH
FBB8- A0 03    5290        LDY #3       (READ 3 BYTES LATER)
FBBA- BD 8C C0 5300 .5     LDA DRV.Q6L,X     GET NEXT BYTE
FBBD- 10 FB    5310        BPL .5
FBBF- C9 96    5320        CMP #$96     BETTER BE...
FBC1- D0 E7    5330        BNE .3       ...IT IS NOT
FBC3- 78       5340        SEI          ...NO INTERRUPTS NOW
FBC4- A9 00    5350        LDA #0       START CHECK SUM
FBC6- 8D 6C FB 5360 .6     STA CHECK.SUM
FBC9- BD 8C C0 5370 .7     LDA DRV.Q6L,X     GET NEXT BYTE
FBCC- 10 FB    5380        BPL .7       1X1X1X1X
FBCE- 2A       5390        ROL          X1X1X1X1
FBCF- 8D 6B FB 5400        STA X1X1X1X1
FBD2- BD 8C C0 5410 .8     LDA DRV.Q6L,X     GET NEXT BYTE
FBD5- 10 FB    5420        BPL .8       1Y1Y1Y1Y
FBD7- 2D 6B FB 5430        AND X1X1X1X1   XYXYXYXY
FBDA- 99 6D FB 5440        STA HDR.CHKSUM,Y
FBDD- 4D 6C FB 5450        EOR CHECK.SUM
FBE0- 88       5460        DEY
FBE1- 10 E3    5470        BPL .6
FBE3- A8       5480        TAY          CHECK CHECKSUM
FBE4- D0 15    5490        BNE .11      NON-ZERO, ERROR
FBE6- BD 8C C0 5500 .9     LDA DRV.Q6L,X     GET NEXT BYTE
FBE9- 10 FB    5510        BPL .9
FBEB- C9 DE    5520        CMP #$DE     TRAILER EXPECTED $DE.AA.EB
FBED- D0 0C    5530        BNE .11      NO, ERROR
FBEF- EA       5540        NOP
FBF0- BD 8C C0 5550 .10    LDA DRV.Q6L,X
FBF3- 10 FB    5560        BPL .10
FBF5- C9 AA    5570        CMP #$AA
FBF7- D0 02    5580        BNE .11      NO, ERROR
FBF9- 18       5590        CLC
FBFA- 60       5600        RTS
FBFB- 38       5610 .11    SEC
FBFC- 60       5620        RTS
               5630 *--------------------------------
               5640 READ.SECTOR
FBFD- 8A       5650        TXA          SLOT*16 ($60 FOR SLOT 6)
FBFE- 09 8C    5660        ORA #$8C     BUILD Q6L ADDRESS FOR SLOT
FC00- 8D 5A FC 5670        STA .9+1     STORE INTO READ-DISK OPS
FC03- 8D 73 FC 5680        STA .12+1
FC06- 8D 89 FC 5690        STA .13+1
FC09- 8D 9D FC 5700        STA .15+1
FC0C- 8D B2 FC 5710        STA .18+1
FC0F- A5 44    5720        LDA RWB.BUFFER    PLUG CALLER'S BUFFER
FC11- A4 45    5730        LDY RWB.BUFFER+1  ADDRESS INTO STORE'S
FC13- 8D AF FC 5740        STA .17+1    PNTR FOR LAST THIRD
FC16- 8C B0 FC 5750        STY .17+2
```

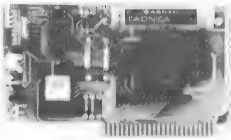
```
FC19- 38          5760        SEC             PNTR FOR MIDDLE THIRD
FC1A- E9 54       5770        SBC #84
FC1C- B0 01       5780        BCS .1
FC1E- 88          5790        DEY
FC1F- 8D 97 FC    5800 .1     STA .14+1
FC22- 8C 98 FC    5810        STY .14+2
FC25- 38          5820        SEC             PNTR FOR BOTTOM THIRD
FC26- E9 57       5830        SBC #87
FC28- B0 01       5840        BCS .2
FC2A- 88          5850        DEY
FC2B- 8D 70 FC    5860 .2     STA .11+1
FC2E- 8C 71 FC    5870        STY .11+2
                  5880 *---FIND $D5.AA.AD HEADER--------
FC31- A0 20       5890        LDY #32         MUST FIND $D5 WITHIN 32 BYTES
FC33- 88          5900 .3     DEY
FC34- F0 37       5910        BEQ .10         ERROR RETURN
FC36- BD 8C C0    5920 .4     LDA DRV.Q6L,X
FC39- 10 FB       5930        BPL .4
FC3B- 49 D5       5940 .5     EOR #$D5
FC3D- D0 F4       5950        BNE .3
FC3F- EA          5960        NOP
FC40- BD 8C C0    5970 .6     LDA DRV.Q6L,X
FC43- 10 FB       5980        BPL .6
FC45- C9 AA       5990        CMP #$AA
FC47- D0 F2       6000        BNE .5
FC49- EA          6010        NOP
FC4A- BD 8C C0    6020 .7     LDA DRV.Q6L,X
FC4D- 10 FB       6030        BPL .7
FC4F- C9 AD       6040        CMP #$AD
FC51- D0 E8       6050        BNE .5
                  6060 *---READ 86 BYTES INTO TBUF...TBUF+85----------
                  6070 *---THESE ARE THE PACKED LOWER TWO BITS--------
                  6080 *---FROM EACH BYTE OF THE CALLER'S BUFFER.-----
FC53- A0 AA       6090        LDY #170
FC55- A9 00       6100        LDA #0          INIT RUNNING EOR-SUM
FC57- 85 3A       6110 .8     STA RUNNING.SUM
FC59- AE EC C0    6120 .9     LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
FC5C- 10 FB       6130        BPL .9
FC5E- BD 00 F9    6140        LDA BYTE.TABLE,X      DECODE DATA
FC61- 99 56 FA    6150        STA TBUF-170,Y
FC64- 45 3A       6160        EOR RUNNING.SUM
FC66- C8          6170        INY
FC67- D0 EE       6180        BNE .8
                  6190 *---READ NEXT 86 BYTES------------------------
                  6200 *---STORE 1ST 85 IN BUFFER...BUFFER+84---------
                  6210 *---SAVE THE 86TH BYTE ON THE STACK------------
FC69- A0 AA       6220        LDY #170
FC6B- D0 05       6230        BNE .12         ...ALWAYS
                  6240 *--
FC6D- 38          6250 .10    SEC             I/O ERROR EXIT
FC6E- 60          6260        RTS
                  6270 *--
FC6F- 99 55 46    6280 .11    STA BUFF.BASE-171,Y
FC72- AE EC C0    6290 .12    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
FC75- 10 FB       6300        BPL .12
FC77- 5D 00 F9    6310        EOR BYTE.TABLE,X      DECODE DATA
FC7A- BE 56 FA    6320        LDX TBUF-170,Y        MERGE LOWER 2 BITS
FC7D- 5D 00 FA    6330        EOR BIT.PAIR.TABLE,X
FC80- C8          6340        INY
FC81- D0 EC       6350        BNE .11
FC83- 48          6360        PHA             SAVE LAST BYTE (LATER BUFFER+85)
                  6370 *---READ NEXT 86 BYTES-----------
                  6380 *---STORE AT BUFFER+86...BUFFER+171-----------
FC84- 29 FC       6390        AND #$FC              MASK FOR RUNNING EOR.SUM
FC86- A0 AA       6400        LDY #170
FC88- AE EC C0    6410 .13    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
FC8B- 10 FB       6420        BPL .13
FC8D- 5D 00 F9    6430        EOR BYTE.TABLE,X      DECODE DATA
FC90- BE 56 FA    6440        LDX TBUF-170,Y        MERGE LOWER 2 BITS
FC93- 5D 01 FA    6450        EOR BIT.PAIR.TABLE+1,X
FC96- 99 AC 46    6460 .14    STA BUFF.BASE-84,Y
FC99- C8          6470        INY
FC9A- D0 EC       6480        BNE .13
                  6490 *---READ NEXT 84 BYTES------------------------
                  6500 *---INTO BUFFER+172...BUFFER+255---------------
FC9C- AE EC C0    6510 .15    LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
FC9F- 10 FB       6520        BPL .15
FCA1- 29 FC       6530        AND #$FC
FCA3- A0 AC       6540        LDY #172
```

```
FCA5- 5D 00 F9 6550 .16      EOR BYTE.TABLE,X      DECODE DATA
FCA8- BE 54 FA 6560          LDX TBUF-172,Y        MERGE LOWER 2 BITS
FCAB- 5D 02 FA 6570          EOR BIT.PAIR.TABLE+2,X
FCAE- 99 00 47 6580 .17      STA BUFF.BASE,Y
FCB1- AE EC C0 6590 .18      LDX DRV.Q6L+MODIFIER  READ NEXT BYTE
FCB4- 10 FB    6600          BPL .18
FCB6- C8       6610          INY
FCB7- D0 EC    6620          BNE .16
FCB9- 29 FC    6630          AND #$FC
               6640 *---END OF DATA-----------------
FCBB- 5D 00 F9 6650          EOR BYTE.TABLE,X      DECODE DATA
FCBE- D0 0C    6660          BNE .20             ...BAD CHECKSUM
FCC0- A6 3E    6670          LDX SLOT.X16        CHECK FOR TRAILER $DE
FCC2- BD 8C C0 6680 .19      LDA DRV.Q6L,X
FCC5- 10 FB    6690          BPL .19
FCC7- C9 DE    6700          CMP #$DE
FCC9- 18       6710          CLC
FCCA- F0 01    6720          BEQ .21             ...GOOD READ!
FCCC- 38       6730 .20      SEC                 ...SIGNAL BAD READ
FCCD- 68       6740 .21      PLA                 STORE BYTE AT BUFFER+85
FCCE- A0 55    6750          LDY #85
FCD0- 91 44    6760          STA (RWB.BUFFER),Y
FCD2- 60       6770          RTS
               6780 *--------------------------------
               6790 UPDATE.TRACK.TABLE
FCD3- 20 F1 FC 6800          JSR GET.SSSD.IN.X
FCD6- 9D 58 FB 6810          STA OLD.TRACK.TABLE,X
FCD9- 60       6820          RTS
               6830 *--------------------------------
               6840 CHECK.IF.MOTOR.RUNNING
FCDA- A6 3E    6850          LDX SLOT.X16
               6860 CHECK.IF.MOTOR.RUNNING.X
FCDC- A0 00    6870          LDY #0
FCDE- BD 8C C0 6880 .1       LDA DRV.Q6L,X       READ CURRENT INPUT REGISTER
FCE1- 20 F0 FC 6890          JSR .2              ...12 CYCLES...
FCE4- 48       6900          PHA                 ...7 MORE CYCLES...
FCE5- 68       6910          PLA
FCE6- DD 8C C0 6920          CMP DRV.Q6L,X       BY NOW INPUT REGISTER
FCE9- D0 05    6930          BNE .2              SHOULD HAVE CHANGED
FCEB- A9 28    6940          LDA #$28      ERROR CODE: NO DEVICE CONNECTED
FCED- 88       6950          DEY           BUT TRY 255 MORE TIMES
FCEE- D0 EE    6960          BNE .1        ...RETURN .NE. IF MOVING...
FCF0- 60       6970 .2       RTS           ...RETURN .EQ. IF NOT MOVING...
               6980 *--------------------------------
               6990 GET.SSSD.IN.X
FCF1- 48       7000          PHA           SAVE A-REG
FCF2- A5 43    7010          LDA RWB.SLOT      DSSSXXXX
FCF4- 4A       7020          LSR
FCF5- 4A       7030          LSR
FCF6- 4A       7040          LSR
FCF7- 4A       7050          LSR           0000DSSS
FCF8- C9 08    7060          CMP #8        SET CARRY IF DRIVE 2
FCFA- 29 07    7070          AND #7        00000SSS
FCFC- 2A       7080          ROL           0000SSSD
FCFD- AA       7090          TAX           INTO X-REG
FCFE- 68       7100          PLA           RESTORE A-REG
FCFF- 60       7110          RTS

               9250 *--------------------------------
               9260 WAIT.FOR.OLD.MOTOR.TO.STOP
FE9B- 4D 59 FB 9270          EOR OLD.SLOT        SAME SLOT AS BEFORE?
FE9E- 0A       9280          ASL                 (IGNORE DRIVE)
FE9F- F0 1C    9290          BEQ .2              ...YES
FEA1- A9 01    9300          LDA #1        LONG MOTOR.TIME
FEA3- 8D 70 FB 9310          STA MOTOR.TIME+1  (COUNTS BACKWARDS)
FEA6- AD 59 FB 9320 .1       LDA OLD.SLOT
FEA9- 29 70    9330          AND #$70
FEAB- AA       9340          TAX
FEAC- F0 0F    9350          BEQ .2        ...NO PREVIOUS MOTOR RUNNING
FEAE- 20 DC FC 9360          JSR CHECK.IF.MOTOR.RUNNING.X
FEB1- F0 0A    9370          BEQ .2        ...NOT RUNNING YET
FEB3- A9 01    9380          LDA #1        DELAY ANOTHER 100 USECS
FEB5- 20 85 FB 9390          JSR DELAY.100
FEB8- AD 70 FB 9400          LDA MOTOR.TIME+1
FEBB- D0 E9    9410          BNE .1        KEEP WAITING
FEBD- 60       9420 .2       RTS
               9430 *--------------------------------
```

APPLIED ENGINEERING
THE BEST PERIPHERALS FOR THE BEST COMPUTER

The TIMEMASTER
Finally a clock that does it ALL!



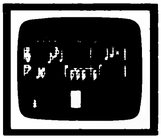
- Designed in 1983 using I.C. technologies that simply did not exist when most other Apple clocks were designed.
- Just plug it in and your programs can read the year, month, date, day, and time to 1 millisecond! The only clock with both year and ms.
- Powerful 2K ROM driver No clock could be easier to use.
- Full emulation of most other clocks, including Mountain Hardware's Appleclock (but you'll like the TIMEMASTER mode better)
- Basic, Machine Code, CP/M and Pascal software on 2 disks!
- Eight software controlled interrupts so you can execute two programs at the same time. (Many examples are included)
- On board timer lets you time any interval up to 48 days long down to the nearest millisecond.

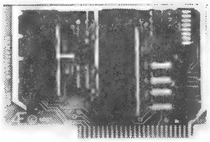
The TIMEMASTER includes 2 disks with some really fantastic time oriented programs (over 25) plus a DOS dater so it will automatically add the date when disk files are created or modified. This disk is over a $200.00 value alone -- we give the software others sell. All software packages for business, data base management and communications are made to read the TIMEMASTER.

If you want the most powerful and the easiest to use clock for your Apple, you want a TIMEMASTER.

PRICE $129.00

Super Music Synthesizer

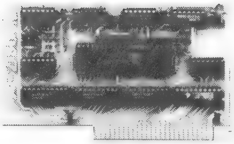




- Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo, boot the disk supplied and you are ready to input and play songs.
- It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.
- Now with new improved software for the easiest and fastest music input system available anywhere.
- We give you lots of software. In addition to Compose and Play programs, 2 disks are filled with over 30 songs ready to play.
- Easy to program in Basic to generate complex sound effects. Now your games can have explosions, phaser zaps, train whistles, death cries. You name it, this card can do it.
- Four white noise generators which are great for sound effects.
- Plays music in true stereo as well as true discrete quadraphonic.
- Full control of attack, volume, decay, sustain and release.
- Will play songs written for ALF synthesizer (ALF software will not take advantage of all the features of this board. Their software sounds the same in our synthesizer.)
- Automatic shutoff on power-up or if reset is pushed.
- Many many more features.

PRICE $159.00

Z-80 PLUS!



- TOTALLY compatible with ALL CP/M software.
- The only Z-80 card with a special 2K "CP/M detector" chip.
- Fully compatible with microsoft disks (no pre-boot required).
- All new 1983 design incorporates the latest in I.C. technologies.

- Red "CP/M WORKING" LED indicator, the Z-80 Plus does not interfere with non-CP/M programs.
- An on-card PROM eliminates many I.C.'s for a cooler, less power consuming board. (We use the Z-80A at a fast 4MHZ)
- Does EVERYTHING the other Z-80 boards do, plus Z-80 interrupts.

Don't confuse the Z-80 Plus with crude copies of the microsoft card. The Z-80 Plus employs a much more sophisticated and reliable design. With the Z-80 Plus you can access the largest body of software in existence. Two computers in one and the advantages of both, all at an unbelievably low price.

PRICE $139.00

COMING SOON: The Z-80 Plus for the Apple III

Viewmaster 80

There used to be about a dozen 80 column cards for the Apple, now there's only ONE.

- TOTALLY Videx Compatible
- 80 characters by 24 lines, with a sharp 7x9 dot matrix
- On-board 40/80 soft video switch with manual 40 column override
- Fully compatible with ALL Apple languages and software -- there are NO exceptions
- Low power consumption through the use of CMOS devices
- All connections on the card are made with standard video connectors, no cables are soldered to the board
- All new 1983 design (using a new Microprocessor based C.R.T. controller)

JUST COMPARE!

		40/80 SOFTSWITCH	SHIFT KEY SUPPORT	PASCAL COMPATIBLE	7x9 DOT MATRIX	LIGHT PEN INPUTS	40 COLUMN OVERRIDE	INVERSE CHARACTERS
VIEWMASTER	**169**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
SUP'RTERM	375	NO	YES	NO	NO	NO	YES	YES
WIZARD80	245	NO	NO	YES	YES	NO	YES	YES
VISION80	375	YES	YES	YES	YES	NO	NO	NO
OMNIVISION	295	NO	YES	NO	NO	NO	YES	YES
VIEWMAX80	219	YES	YES	YES	YES	NO	NO	YES
SMARTERM	360	YES	YES	YES	NO	NO	YES	NO
VIDEOTERM	345	NO	NO	NO	YES	YES	NO	YES

The VIEWMASTER 80 works with all 80 column applications including CP/M, Pascal, WordStar, Format II, Easywriter, Apple Writer II, Viscalc, and many others. The VIEWMASTER 80 is THE MOST compatible 80 column card you can buy at ANY price!

PRICE $169.00

MemoryMaster IIe 128K RAM Card

- Expands your Apple IIe to 192K memory
- Provides an 80 column text display
- Compatible with all Apple IIe 80 column and extended 80 column card software (Same physical size as Apple's 64K card)
- Available in 64K and 128K configurations
- Bank select LED's for each 64K bank
- Permits your IIe to use the new double high resolution graphics
- Automatically expands Visicalc to 95K storage in 80 columns! The 64K configuration is all that's needed, 128K can take you even higher.

- Complete documentation included, we show you how to use all 128K.

If you already have Apple's 64K card, just order the MEMORYMASTER with 64K and use the 64K from your old board to give you a full 128K. (The board is fully socketed so you simply plug in more chips.)

MemoryMaster with 128K	**$249**
Upgradeable MemoryMaster with 64K	**$169**
Non-Upgradeable MemoryMaster with 64K	**$149**

Our boards are far superior to most of the consumer electronics made today. All I.C.'s are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products work in APPLE IIe, II, II+ and Franklin (except MemoryMaster). Applied Engineering also manufactures a full line of data acquisition and control products for the Apple; A/D converters and digital I/O cards, etc. Please call for more information. All our products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle **THREE YEAR WARRANTY.**

All Orders Shipped Same Day. Texas Residents Add 5% Sales Tax. Add $10.00 If Outside U.S.A. Dealer Inquiries Welcome.

Send Check or Money Order to
APPLIED ENGINEERING
P.O. Box 470301
Dallas, TX 75247

Call (214) 492-2027
7a.m. to 11p.m. 7 days a week
MasterCard, Visa & C.O.D. Welcome

Qwerty 68000 Training/Development System...Bob Sander-Cederlof

There is now a plethora of 68000 boards designed to fit inside, or nearly inside, your Apple. Names like DTACK Grounded, PDQ, Saybrook, and Acorn.

Most of these are aimed at hot-rodding your Apple. Some come with the UCSD p-System, including Pascal and an Applesoft-compatible BASIC and much more. Others have a more limited selection. Most are too costly for most of us, around $1500.

Motorola and others sell development systems based on the 68000 for $10K-30K. The Apple Lisa makes an excellent development system, at $6995 plus the developer's software kit (when it becomes available).

> "Wait a minute! I don't even have a spare $1500, let alone $10K! And I want to get my feet wet first, before diving in over my head!"
>
> "In fact, I want to try my hand at learning 68000 assembly language first. I need an assembler, some books, and a monitor with step and trace commands. I would like a hands-on tutorial I can work though at my own pace."
>
> "I can't afford to lay out more than $750 right now. But I want an expandable system, that can grow with my knowledge and needs."

Guess what...somebody overheard our thoughts! Jerry Hansen and Lane Hauck, of Qwerty Inc., have put together a package deal too good to resist: a complete integrated training and software development package for only $695.

The package includes a card to plug in any slot of your Apple II, II Plus, or //e; a reference manual which leads you through the details of the card, their firmware, and the assembler; a full-fledged macro assembler; the best three reference books, with other booklets and reference cards. You can use the books in a hands-on tutorial fashion, mastering the 68000 assembly language as you go.

The Q-68 card is the heart of the package. It is a compact, well-crafted design, with a 68008 microprocessor, 2K bytes of RAM, and 8K bytes of EPROM. The full Apple address-space can be addressed by the 68008 as well, including any memory expansion cards you may have. RAM can be expanded on-board to 8K, and EPROM to 32K. A 50-pin expansion connector allows connection of additional memory, to a total of 1 megabyte.

You don't need any external power supply or chassis. The card draws a maximum of 400 milliamps. (While this is more than Apple will recommend, it seems to be well within the capability of the Apple power supply.) If you don't already have a cooling fan, you will probably want one after installing this card. The 68008 is the main power user, which fact makes me ever-so-hungry for a CMOS version.

The 68008 is a trimmed-down version of the 68000, with an 8-bit data bus. The instruction set is unchanged, but it comes in a smaller package: fewer pins, fewer milliamps, fewer dollars. On the Q-68 board, the 68008 is clocked at 7.16 MHz.

The Apple 6502 keeps running while the 68008 is executing code; when the 68008 refers to Apple memory, the 68008 slows down to wait for the Apple bus, and the Apple slows to half speed during that cycle. True multiprocessing is possible.

The Q-68 EPROM is loaded with good things. You get a comprehensive self-test facility, and an easy-to-use debugging monitor. The debugging monitor allows you to step and trace through your programs, and set breakpoints. There are five different display windows you can cycle through with a single keystroke: Register, Memory, Disassembly, and Breakpoint displays, and a helpful Command Summary.

Qwerty is aiming primarily at the those of us who want to learn 68000 programming and/or develop 68000 software without investing in an expensive complete 68000 system. However, there are many other exciting possibilities for this board. Those of you who really do want to speed up your Apple can certainly write code for the purpose. (Or maybe adapt public domain code already written for other 68K boards.) The Q-68 card may be used as a powerful controller or co-processor with your yet-to-be-written software. You can connect the Q-68 to the outside world directly, as well as through the Apple bus.

Now for something truly unique: the package comes with a special version of the S-C 68000 Cross Assembler. The S-C manual has been re-written to give 68000 code examples throughout. New commands have been added to start the Q-68 card, either in debug mode or at full speed. Three versions are included to provide different memory usage options.

What you get is a near optimum environment both for learning and for serious software development. Gone are the "load the editor, load-edit-save the source program, load the assembler, assemble, load the loader, load the object program, run into a bug, load the editor...." blues. With this package you simply edit, assemble, and run directly from RAM.

Programs too large for RAM can be assembled and loaded using multiple source and object files when necessary, but you still never need to reload the editor/assembler or monitor/debugger.

Current users of the S-C Assembler family already know the commands and editing techniques. You can concentrate on learning the 68000 itself, and the Qwerty debugger, without being distracted by a whole new operating system. (Later, when you can afford a Lisa or MacIntosh, you will already know the language and can concentrate on learning the operating system.)

Here is another new twist: Qwerty offers a free 30-day trial period. If you're not happy with the package for any reason, you can return it within 30 days in salable condition for a full refund. Qwerty, Inc. Phone (619) 569-5283.

A Look at the Aztec C Compiler for Apple DOS........Bill Morgan

As I mentioned last month, I'm getting very interested in the C language. That August issue of Byte definitely turned me on, so I've started to look at ways to get C into my Apple.

Byte featured a comparative review of several C compilers for CP/M. One of the highest-rated was the Aztec C Compiler System, which is also available for Apple DOS 3.3. The Aztec compiler was given especially high marks for being truly complete and compatible with the standard for C, the book "The C Programming Language", by Kernigan and Ritchie.

I haven't had a chance to actually do any programming with the Aztec system yet, but, thanks to Donna Lamb, a subscriber in New York City, I was able to spend an afternoon looking over the manual. Here are some of my impressions.

Manual

The manual is 135 pages long in 5 chapters and 2 appendices:

Tutorial Intro - 15pp - Getting started, configuring and using the SHELL, compiling, assembling, linking and executing. A get-your-toes-damp intro to the system.

Shell - 22pp - The SHELL program resides in the language card, at $D000-$F7FF. It replaces the Command Interpreter portion of DOS 3.3 and provides a UNIX-like user interface, including I/O redirection and command parsing with argument passing.

Programs - 23pp - Using the editor, compilers, assemblers, linker, and utilities.

Libraries - 33pp - Discussion of the Standard I/O, System I/O, Utility, and Math Routines supplied with the system.

Technical Info - 28pp - Miscellaneous information on the internals of the system and the assembly-language interface. Manx promises continuing additions to this chapter, as part of the updates.

Appendices - 12pp - Error messages and examples of the compiler and assembler outputs for a simple program.

DOS 3.3 Interface

The disks you receive from Manx do not include DOS, so to enter the system you must first boot DOS, then BRUN SHELL.

SHELL overlays the DOS Command Interpreter and patches at least two (unspecified) points inside the File Manager. All the documentation has to say about non-standard (i.e., fast) DOS's is "try it and see." I am told that Diversi-DOS does not work; I don't know about others.

DOWNLOADING
CUSTOM CHARACTER SETS

One of the features 'hidden' in many printers available today is their ability to accept user-defined character sets. With the proper software, these **custom characters** are 'downloaded' from your Apple II computer to the printer in a fraction of a second. Once the printer has 'learned' these new characters, they will be remembered until the printer is turned off.

After the downloading operation, you can use your printer with virtually any word processor. Just think of the possibilities! There's nothing like having your own **CUSTOM CHARACTERS** to help convey the message. And you still have access to those built-in fonts as well! **Here's a quick look at some possible variations:**

	BUILT-IN	CUSTOM
10CPI:	AaBbCcDdEeFfGgHhIiJjKk	**AaBbCcDdEeFfGgHhIiJjKk**
12CPI:	AaBbCcDdEeFfGgHhIiJjKk	**AaBbCcDdEeFfGgHhIiJjKk**
17CPI:	AaBbCcDdEeFfGgHhIiJjKk	**AaBbCcDdEeFfGgHhIiJjKk**
5CPI:	AaBbCcDdEeFf	**AaBbCcDdEeFf**
6CPI:	AaBbCcDdEeFf	**AaBbCcDdEeFf**
8CPI:	AaBbCcDdEeFf	**AaBbCcDdEeFf**

And let's not forget Enhanced and Underined printing as well...

AaBbCcDdEeFfGgHhIiJjKk	**AaBbCcDdEeFfGgHhIiJjKk**
AaBbCcDdEeFfGgHhIiJjKk	**AaBbCcDdEeFfGgHhIiJjKk**

The Font Downloader & Character Editor software package has been developed by RAK-WARE to help you unleash the power of your printer. The basic package includes the downloading software with 4 fonts to get you going. Also included is a character editor so that you can turn your creativity loose. Use it to generate unique character fonts, patterns, symbols and graphics. A detailed user's guide is provided on the program diskette.

SYSTEM REQUIREMENTS:

* APPLE II, APPLE II Plus, APPLE //e or lookalike with 48K RAM
* 'DUMB' Parallel Printer Interface Board (like Apple's Parallel Printer Interface, TYMAC's PPC-100 or equivalent)

The Font Downloader & Editor package is only $39.95 and is currently available for either the Apple Dot Matrix Printer or C.Itoh 8510AP (specify printer). Epson FX-80 and OkiData versions coming soon. Enclose payment with order to avoid $3.00 handling & postage charge.

RAK-WARE
41 Ralph Road West Orange New Jersey 07052

Say You Saw It In **APPLE ASSEMBLY LINE**!

Two Compilers for the Price of One

The Aztec system includes two separate compilers and two assemblers. There is a compiler/assembler pair for generating native 6502 code, and another compiler/assembler for an interpreted pseudo-code. The native code is fast but large, while the pseudo-code is slower but smaller. You can compile most of your program to pseudo-code, compile the time-critical parts to machine code, and write any extremely critical sections directly in assembly language. You can then link all these different object modules into one executable program.

Updates

The copy I saw was Version 1.05b of the Aztec system. Updates are available for an unspecified "nominal" fee, or an automatic update service is available for $50 per year.

Drawbacks

The people I have talked to who use the Aztec system regularly mention two drawbacks: compilation time and program size. Much of the compile time problem seems to be a matter of the Apple's disk speed, which can be improved.

The program size is related to the size of the run-time routines and the libraries included in a program. Experienced C programmers say that it is usually possible to manipulate the libraries to minimize the size of included code, but that is a fairly advanced technique.

ProDOS Version

There is supposed to be a ProDOS version of the Aztec system, which should be significantly faster, coming sometime. It's too soon to tell when that is likely to appear, so we'll just have to wait. The ProDOS version will be marketed as a completely separate version, rather than as an update to the DOS 3.3 version.

Conclusions

The Aztec C Compiler System is a full C compiler that runs in an Apple][, and that makes it unique. Since my interest is in learning C and starting to develop programs that will be used on other, more powerful computers, I plan to place my order as soon as the ProDOS version is available.

All things considered, the Aztec system is not a great approach for developing applications intended only for use on Apple][computers. The Apple is simply too limited for full C.

$199, from: Manx Software Systems, Box 55, Shrewsbury, NJ 07701. (201) 780-4004.

Hitachi 6301 Cross Support................Bob Sander-Cederlof

As you probably know, we have a growing line of cross assemblers available. You can use your Apple as a development system without ever learning another editor/assembler/operating-system, on any of ten or more different chips.

It all started back in 1980 when Nigel Nathan paid me to create a 6801 cross assembler based on version 4.0 of the S-C Assembler II. Later Bob Urschel bought a copy. Back then we thought $300 a copy was a pretty good price.

All our competition in this field seems to agree. Avocet charges $200 or more per cross assembler. Byte magazine carries several ads showing prices for cross assemblers between $395 and $1000 apiece. Our assemblers are just as good, and many of you tell us ours are easier to use and more powerful. But we charge either $32.50 or $50 apiece, after you own the $80 S-C Macro Assembler.

Until very recently, the 6800/1/2 Macro Cross Assembler came with only one version on the disk. This one version assembled all of the opcodes of the 6801 chip. If you were programming for a 6800, which did not support all of those opcodes and addressing modes, it was a little dangerous. Last month we upgraded this disk by making two versions: one for 6800 only, and one for 6801.

Now I have added a third version for the Hitachi 6301. The 6301 is a CMOS chip, includes all the opcodes of the 6801, and adds six more:

XGDX	Exchange D and X
SLP	Sleep (reduced power mode)
AIM	And Immediate into Memory
OIM	Or Immediate into Memory
EIM	Exclusive Or Immediate into Memory
TIM	Test Memory Immediate

The last four each have two addressing modes. You can write "AIM #val,addr" or "AIM #val,addr,X". In both modes the address is only 8 bits. You can see that AIM lets you clear any bits in a memory byte; OIM lets you set any bits in a byte; EIM lets you toggle any bits; and TIM lets you test any bits. TIM forms the logical product (AND) of the memory byte and the immediate value, and tests for sign and zero.

The 6301 includes extensive memory mapped I/O on the chip, mapped into the zero page. With these "xIM" opcodes you have an extremely powerful I/O capability.

If you have the older disk of the 6800/1/2 cross assembler, and want to upgrade to get the 6301 version, send $5.

Killing the EXEC.....................................Bob Bragner

Istanbul, Turkey

Have you ever been at the beginning of the execution of a l-o-n-g EXEC file and realized you didn't really want to go through with it? There's not really much you can do. Control-C and RESET are ineffective even if you have an old Apple][without the Autostart ROM. On a //e you can hit Control-Open Apple-RESET, but at the expense of anything you may have in the Apple's memory -- a rather drastic solution.

As it turns out, there is a very easy way to terminate an EXEC file in progress. Apple DOS contains a single byte ($AAB3 when DOS is at its normal location) which is called "EXEC.STATUS". If the value of this byte is not 0 DOS thinks an EXEC file is in charge. If it is 0 then as far as DOS is concerned, no EXEC file is active. So we have the following little routine:

```
                  1000 *SAVE S.KILL.EXEC
                  1010 *--------------------------------
03F2-             1020 RESET        .EQ $3F2
FB6F-             1030 SET.PWR.BYTE .EQ $FB6F
03D0-             1040 DOS.ENTRY    .EQ $3D0
AAB3-             1050 EXEC.STATUS  .EQ $AAB3
                  1060 *--------------------------------
                  1070        .OR $300
                  1080        .TF B.KILL.EXEC
                  1090 *--------------------------------
0300- A9 0D       1100 INIT   LDA #KILL.EXEC
0302- 8D F2 03    1110        STA RESET
0305- A9 03       1120        LDA /KILL.EXEC
0307- 8D F3 03    1130        STA RESET+1
030A- 4C 6F FB    1140        JMP SET.PWR.BYTE
                  1150 *--------------------------------
                  1160 KILL.EXEC
030D- A9 00       1170        LDA #0
030F- 8D B3 AA    1180        STA EXEC.STATUS
0312- 4C D0 03    1190        JMP DOS.ENTRY
```

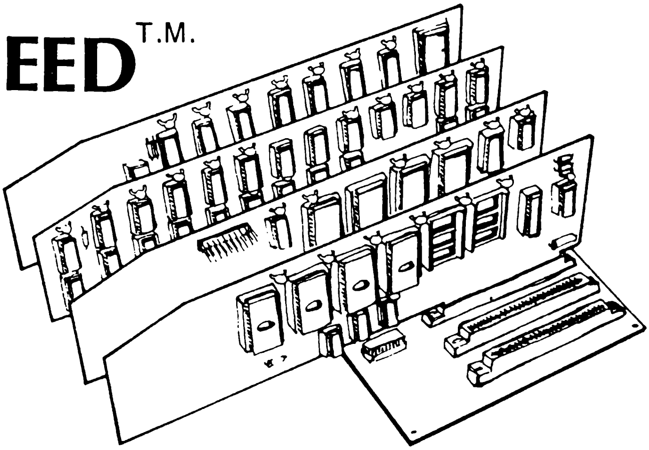
This routine can be reassembled to run anywhere. the INIT portion simply directs the RESET vector to the KILL.EXEC part of the routine and must be called before the EXEC command is issued. KILL.EXEC stores a 0 in the EXEC.STATUS flag and jumps to the DOS warm start at $3D0. Now if you hit RESET during an EXEC file's operation, the file will terminate politely.

Here is a series of POKES and a CALL that could be placed at the beginning of any EXEC program:

```
     POKE 1010,13 : POKE 1011,3 : CALL 64367
     POKE 781,169 : POKE 782,0 : POKE 783,141 : POKE 784,179
     POKE 785,170 : POKE 786,76 : POKE 787,208 : POKE 788,3
     (the rest of your program goes here)
```

This works from machine language, Integer BASIC, Applesoft, AND the S-C RAMcard Macro Assembler. The latter is a big help when you discover you're EXEC'ing the wrong 2000-line text file into the assembler, or you've forgotten to turn AUTO on!

[Just a couple of comments: this trick won't work with an old non-Autostart ROM Apple, since you can't redirect RESET; and be sure to type the CLOSE command after the RESET, to free up the file buffer that the EXEC file was using.   Bill]



The Computer Hacker and Dataphile Digest

I received Vol 1 No 2 of the "Computer Hacker", and I think it will be a useful newsletter. As the magazines become more and more general, filled with reviews of packaged systems and software, we will have to look elsewhere for articles that get down to the nitty-gritty. Even local club newsletters are steering away from the hobbyist's or technician's needs.

The issue I have includes listings of a pair of programs to transfer data from one computer to another in the CP/M environment; part two of a detailed explanation of the RS-232 "standard"; part one of directions for building a hardware print spooler; a review of floppy disk formats; an Apple (6502) assembly language program for sending Morse code; and a beginner's introduction to electronics.

The Computer Hacker, 12 issues per year for $24, P. O. Box 1697, Kalispell, MT 59903.

Dataphile Digest is a monthly survey of Apple related periodicals. Bill & Shannon Bailey scan more than a dozen magazines each month, and write brief descriptions of each article relating to Apple computers. They organize the descriptions into categories that make it easy to find any topic you like. The second issue covered one or two issues of 14 different magazines, and included 840 entries organized into 38 categories.

Dataphile Digest is typeset, and printed the same size as Apple Assembly Line. The current issue is 78 pages (plus cover and contents pages), and bears a cover price of $3.50. No subscription price is given, so I would suggest writing to them at P. O. Box 2806, Del Mar, CA 92014. Or call at (619) 436-9382.

Shapemaker Enhancements....................Bob Sander-Cederlof

Frank Belanger sent me a new updated version of his Shapemaker Utility. He says it is now the best program of its type on the market, and he is really proud of it. Here are the new features:

* Clearer, more accessible HELP screens.
* RENUMBER command in the Shape Editor.
* Two grid sizes: 18x30 and 24x40.
* Hi-Res Dump for Epson printer, accessible both in Shapemaker and with an &-command.
* Four new typefaces (total now 9).
* Manual now 55 pages long.

Shapemaker is still just $35, from Frank at 4200 Avenue B, Austin, TX 78751.

ProDOS and Clock Drivers, with a..........Bob Sander-Cederlof
Commented Listing of ProDOS $F142-$F1BE

ProDOS is a new operating system which Apple expects to release to the public during the first quarter of 1984. I am told that new computers and disk drives will be shipped with ProDOS rather than DOS 3.3. Version 1.0 is already available to licensed developers (I have it).

Apple has released massive amounts of documentation to licensed developers, and has even been offering a full day class at $225 per seat in various cities around the country. I attended the Dallas class on October 21st. Even with all the help they are giving, there are still a lot of unclear details that can only be illuminated by well-commented assembly listings of the actual ProDOS code. Apple will never publish these, so we will do it ourselves.

My first serious foray into ProDOS began at the request of Dan Pote, Applied Engineering. Dan wanted me to modify the firmware of his Timemaster clock card so that it automatically had full compatibility with ProDOS. Dan wanted all programs, even protected ones, which boot under ProDOS, to be able to read the date and time from his card. Also, he wanted ProDOS to time/date stamp the files in the directory with his card, just as it does with Thunderclock. (No small task, it turned out.)

ProDOS, when booting, searches the slots for a Thunderclock. If it finds one, it marks a bit in the machine ID byte (MACHID, bit 0 of $BF98 = 1); it plugs two bytes at $F14D and F150 with $CN, where N is the slot number; and it stores a JMP opcode ($4C) at $BF06.

$BF06 is a standard vector to whatever clock routine is installed. If no Thunderclock was found, an RTS opcode will be stored there.

The ProDOS boot slot search looks for these Thunderclock ID bytes:

```
       $CN00 = $08
       $CN02 = $28
       $CN04 = $58
       $CN08 = $70
```

After booting, ProDOS loads and executes the program called STARTUP. The standard STARTUP program searches the slots for various cards and displays a list of what it finds. Unfortunately this list seldom agrees with the true configuration in any of my computers. For one thing, STARTUP examines different bytes than the boot search does. In looking for a clock card, STARTUP wants:

```
       $CN00 = $08
       $CN01 = $78
       $CN02 = $28
```

If you do not have a Thunderclock, but do have some other clock, you have several options. What I did for Dan was change the firmware of Timemaster so that it emulates Thunderclock. ProDOS is convinced it has a Thunderclock, but you are saved the extra expense, and you gain extra features.

Another approach is to write a program which installs your own clock driver inside ProDOS. Mike Owen, of Austin, Texas, did this for Dan. After ProDOS boots it loads the first type SYS file it can find in the directory whose name ends with ".SYSTEM". Normally this is "BASIC.SYSTEM", which then proceeds to execute STARTUP. However, you can set up your disk with CLOCK.SYSTEM before BASIC.SYSTEM in the directory.

Write CLOCK.SYSTEM so that it begins at $2000, because all type SYS files load there. The program should mark the clock ID bit in MACHID, punch a JMP opcode at $BF06, and look at the address in $BF07,BF08. That address is the beginning of the clock driver inside the language card. Right now that address is $F142, but it could change.

Your program should write enable the language card by two "LDA $C081" instructions in a row, and then copy your clock driver into the space starting at that address. You can use up to 124 bytes. If your driver has references to the clock slot, be sure to modify them to the actual slot you are using. If your driver has internal references, be sure to modify them to point to the actual addresses inside the new physical location.

It is standard practice in peripheral firmware to use the following code to find out which slot the card is in:

```
       JSR $FF58     A Guaranteed $60 (RTS opcode)
       TSX           Stack pointer
       LDA $100,X    Get $CN off stack
```

Many cards also use "BIT $FF58" as a means for setting the V-bit in the status register. BE AWARE THAT ProDOS DOES NOT HAVE $60 AT $FF58 in the language card!!!!

The Thunderclock has two entries, at $CN08 and $CN0B, which assume that $CN is already in the X-register. $CN0B allows setting the clock mode, and $CN08 reads the clock in the current mode. The ProDOS driver calls on these two entries, as the following listing shows.

ProDOS maintains a full page at $BF00 called the System Global Page. The definition of this page should not change, ever. They say. Locations $BF90-BF93 contain the current date and time in a packed format. A system call will read the clock, if a driver is installed, and format the year-month-day-hour-minute into these four bytes.

Now here is a listing of the current Thunderclock driver, as labelled and commented by me.

```
                  1000 *SAVE S.PRODOS $F142...$F1BE
                  1010 *--------------------------------
                  1020 *  IF THE PRODOS BOOT RECOGNIZES A THUNDERCLOCK,
                  1030 *  A "JMP $F142" IS INSTALLED AT $BF06 AND
                  1040 *  THE SLOT ADDRESS IS PATCHED INTO THE FOLLOWING
                  1050 *  CODE AT SLOT.A AND SLOT.B BELOW.
                  1060 *--------------------------------
BF90-             1070 DATE   .EQ $BF90      $BF91 = YYYYYYYM
                  1080 *                     $BF90 = MMMDDDDD
BF92-             1090 TIME   .EQ $BF92      $BF93 = 000HHHHH
                  1100 *                     $BF92 = 00MMMMMM
0538-             1110 MODE   .EQ $5F8-$C0 THUNDERCLOCK MODE IN SCREEN HOLE
                  1120 *--------------------------------
                  1130        .OR $F142
                  1140        .TA $800
                  1150 *--------------------------------
                  1160 PRODOS.THUNDERCLOCK.DRIVER
F142- AE 50 F1    1170        LDX SLOT.B    $CN
F145- BD 38 05    1180        LDA MODE,X    SAVE CURRENT THUNDERCLOCK MODE
F148- 48          1190        PHA
F149- A9 A3       1200        LDA #$A3      SEND "#" TO THUNDERCLOCK TO
F14B- 20 0B C2    1210        JSR $C20B        SELECT INTEGER MODE
F14D-             1220 SLOT.A .EQ *-1
                  1230 *--------------------------------
                  1240 *      READ TIME & DATE INTO $200...$211 IN FORMAT:
                  1250 *--------------------------------
F14E- 20 08 C2    1260        JSR $C208
F150-             1270 SLOT.B .EQ *-1
                  1280 *--------------------------------
                  1290 *      CONVERT ASCII VALUES TO BINARY
                  1300 *      $3E -- MINUTE
                  1310 *      $3D -- HOUR
                  1320 *      $3C -- DAY OF MONTH
                  1330 *      $3B -- DAY OF WEEK
                  1340 *      $3A -- MONTH
                  1350 *--------------------------------
F151- 18          1360        CLC
F152- A2 04       1370        LDX #4
F154- A0 0C       1380        LDY #12       POINT AT MINUTE
F156- B9 00 02    1390 .1     LDA $200,Y    TEN'S DIGIT
F159- 29 07       1400        AND #$07      IGNORE TOP BIT
F15B- 85 3A       1410        STA $3A       MULTIPLY DIGIT BY TEN
F15D- 0A          1420        ASL           *2
F15E- 0A          1430        ASL           *4
F15F- 65 3A       1440        ADC $3A       *5
F161- 0A          1450        ASL           *10
F162- 79 01 02    1460        ADC $201,Y    ADD UNIT'S DIGIT
F165- 38          1470        SEC
F166- E9 B0       1480        SBC #$B0      SUBTRACT ASCII ZERO
F168- 95 3A       1490        STA $3A,X     STORE VALUE
F16A- 88          1500        DEY           BACK UP TO PREVIOUS FIELD
F16B- 88          1510        DEY
F16C- 88          1520        DEY
F16D- CA          1530        DEX           BACK UP TO PREVIOUS VALUE
F16E- 10 E6       1540        BPL .1        ...UNTIL ALL 5 FIELDS CONVERTED
                  1550 *--------------------------------
                  1560 *      PACK MONTH AND DAY OF MONTH,
                  1570 *--------------------------------
F170- A8          1580        TAY           MONTH (1...12)
F171- 4A          1590        LSR           00000ABC--D
F172- 6A          1600        ROR           D00000AB--C
F173- 6A          1610        ROR           CD00000A--B
F174- 6A          1620        ROR           BCD00000--A
F175- 05 3C       1630        ORA $3C       MERGE DAY OF MONTH
F177- 8D 90 BF    1640        STA DATE      SAVE PACKED DAY AND MONTH
F17A- 08          1650        PHP           SAVE TOP BIT OF MONTH
                  1660 *--------------------------------
                  1670 *      CONVERT MONTH, DAY OF MONTH,
                  1680 *      AND DAY OF WEEK INTO YEAR.
                  1690 *--------------------------------
F17B- 29 1F       1700        AND #$1F      ISOLATE DAY OF MONTH (1...31)
                  1710 *      CARRY SET FOR MONTHS 8...12
F17D- 79 AB F1    1720        ADC YEAR.DAY,Y    COMPUTE DAY OF YEAR
F180- 90 02       1730        BCC .2
F182- 69 03       1740        ADC #3    ADJUST REMAINDER FOR YEARDAY > 255
F184- 38          1750 .2     SEC           GET REMAINDER MODULO 7
F185- E9 07       1760 .3     SBC #7
F187- B0 FC       1770        BCS .3        ...UNTIL ALL 7'S REMOVED
```

```
F189- 69 07     1780        ADC #7       RESTORE TO POSITIVE VALUE
F18B- E5 3B     1790        SBC $3B      SUBTRACT KNOWN DAY OF WEEK
F18D- B0 02     1800        BCS .4       NO BORROW
F18F- 69 07     1810        ADC #7       BORROWED, SO ADD 7 BACK
F191- A8        1820 .4     TAY          ADJUSTED DAY OW WEEK AS INDEX
F192- B9 B8 F1  1830        LDA YRTBL,Y  GET YEAR (82...87)
F195- 28        1840        PLP          GET HIGH BIT OF MONTH IN CARRY
F196- 2A        1850        ROL          FORM YYYYYYYM
F197- 8D 91 BF  1860        STA DATE+1
F19A- A5 3D     1870        LDA $3D      GET HOUR
F19C- 8D 93 BF  1880        STA TIME+1
F19F- A5 3E     1890        LDA $3E      GET MINUTE
F1A1- 8D 92 BF  1900        STA TIME
F1A4- 68        1910        PLA          RESTORE THUNDERCLOCK MODE
F1A5- AE 50 F1  1920        LDX SLOT.B   GET $CN FOR INDEX
F1A8- 9D 38 05  1930        STA MODE,X
F1AB- 60        1940        RTS
                1950 *--------------------------------
F1AB-           1960 YEAR.DAY   .EQ *-1  OFFSET BECAUSE INDEX 1...12
F1AC- 00 1F 3B
F1AF- 5A        1970     .DA #0,#31,#59,#90       JAN,FEB,MAR,APR
F1B0- 78 97 B5
F1B3- D3        1980     .DA #120,#151,#181,#211 MAY,JUN,JUL,AUG
F1B4- F2 14 33
F1B7- 51        1990     .DA #242,#20,#51,#81    SEP,OCT,NOV,DEC
                2000 *--------------------------------
F1B8- 54 54 53
F1BB- 52 57 56
F1BE- 55        2010 YRTBL  .DA #84,#84,#83,#82,#87,#86,#85
                2020 *--------------------------------
```

Lower Case Titles Revisited..........................Bill Morgan

Last month we published Bob Matzinger's patch to Version 1.1 of the Macro Assembler to allow lower-case characters in a .TItle line. The article contained this sentence: "Here is a hex dump of the code, with a square around the byte to be changed:" But I forgot to draw the square on the page!

Here is that section of code again, this time with the square drawn in:

```
A2 00      LDX #0
20 3E x2   JSR $123E or $D23E
C9 2C      CMP #$2C
D0 0D      BNE ...

20 3E x2   JSR $123E or $D23E

B0 08      BCS ...
9D 70 01   STA $170,X
```

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).